ТЕХНИКИ (ТАКТИКИ) ЗА ПОСТИГАНЕ НА КАЧЕСТВОТО НА СОФТУЕРА

Част 2

Преговор

- Дотук разгледахме следните две основни групи тактики
  - Тактики за постигане на изправност (dependability)
    - Основен момент - Репликация на модули
  - Тактики за производителност (performance)
    - Основен момент - Подходяща декомпозиция на модулите

Тактики за изменяемост (*modifiability*)

- Тактиките за постигане на изменяемост също се разделят на няколко групи, в зависимост от техните цели
  - ***Локализиране на промените*** – целта е да се намали броят на модулите, които са директно засегнати от дадена промяна
  - ***Предотвратяване на ефекта на вълната*** – целта е модификациите, необходими за постигането на дадена промяна, да бъдат ограничени само до директно засегнатите модули
  - ***Отлагане на свързването*** – целта е да се контролира времето за внедряване и себестойността на промяната

ТАКТИКИ ЗА ИЗМЕНЯЕМОСТ – ЛОКАЛИЗИРАНЕ НА ПРОМЕНИТЕ

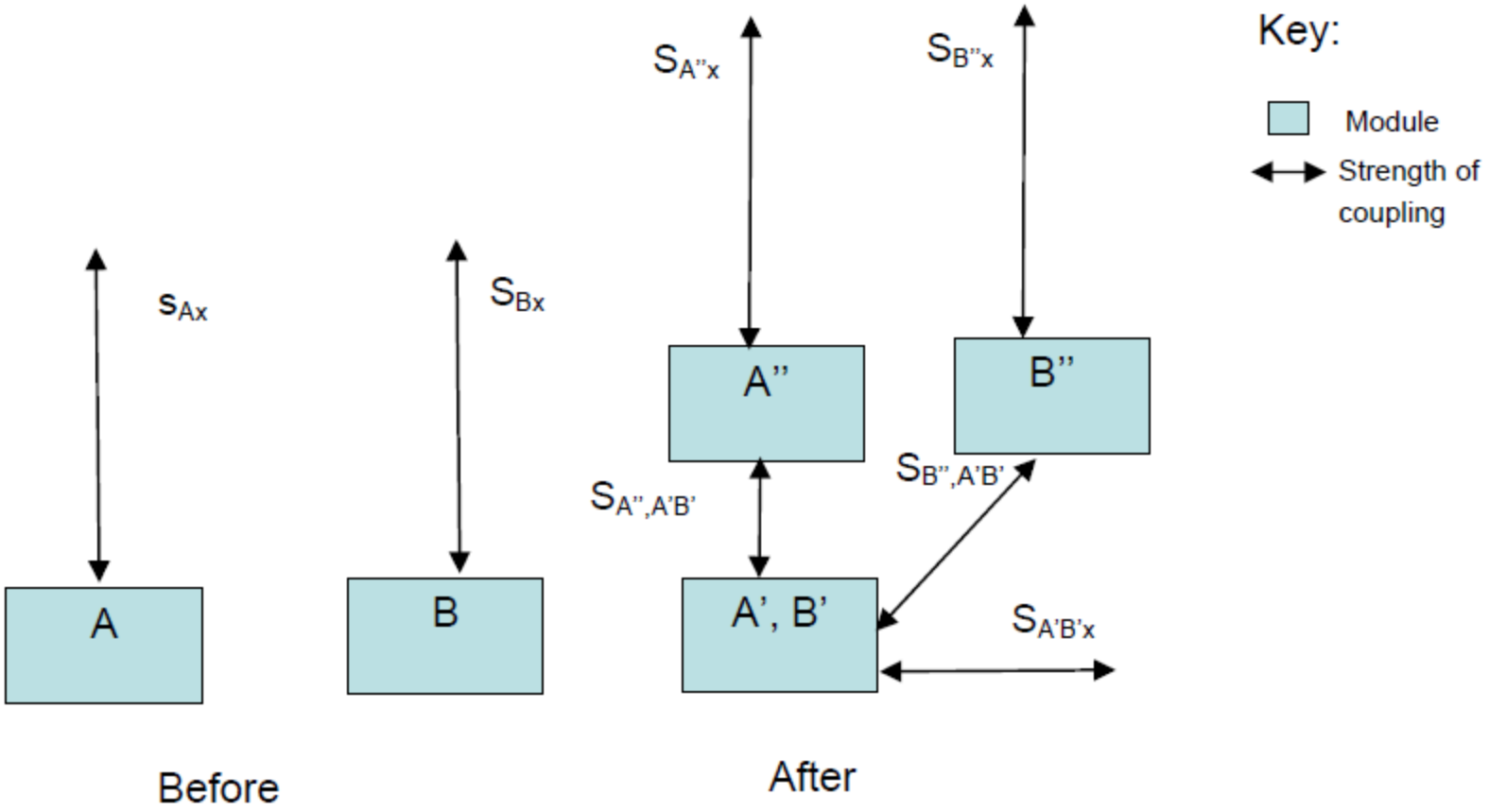
Локализиране на промените

- Въпреки, че няма пряка връзка между броя на модулите, които биват засегнати от дадена промяна и себестойността на извършване на промените, е ясно, че ако промените се ограничат във възможно най-малък брой модули, цената ще намалее.
- Целта на тази група тактики е отговорностите и задачите да бъдат така разпределени между модулите, че обхватът на очакваните промени да бъде ограничен.

Локализиране на промените

- **Поддръжка на семантична свързаност** – семантичната свързаност (semantic coherence) се отнася до отношенията между отговорностите в рамките на даден модул.
- Целта е задачите да се разпределят така, че тяхното изпълнение и реализация да не зависят прекалено много от други модули.
- Постигането на тази цел става като се обединят в рамките на един и същ модул функционалности, които са семантично свързани, при това разглеждани в контекста на очакваните промени.
- Пример за подход в тази насока е и използването на общи услуги (напр. чрез използването на стандартизирани application frameworks или middleware);

Семантична свързаност



Bachmann, F., L. Bass and R. Nord. Modifiability Tactics. SEI Technical Report. September 2007

Локализиране на промените

- **Очакване на промените** – прави се списък на най-вероятните промени (това е трудната част). След което, за всяка промяна се задава въпроса “Помага ли така направената декомпозиция да бъдат локализирани необходимите модификации за постигане на промяната?”.
- Друг въпрос, свързан с първия е “Случва ли се така, че фундаментално различни промени да засягат един и същ модул?”
- За разлика от поддръжката на семантична свързаност, където се очаква промените да са семантично свързани, тук се набляга на конкретните най-вероятни промени и ефектите от тях.
- На практика двете тактики се използват заедно, тъй като списъкът с най-вероятни промени никога не е пълен; доброто обмисляне и съставянето на модули на принципа на семантичната свързаност в много от случаите допълва така направения списък;

Локализиране на промените

- **Ограничаване на възможните опции** – промените, могат да варират в голяма степен и следователно да засягат много модули.
- Ограничаването на възможните опции е вариант за намаляване на този ефект.
  - Напр., вариационна точка в дадена фамилия архитектури (продуктова линия – product line) може да бъде конкретното CPU. Ограничаването на смяната на процесори до тези от една и съща фамилия е възможна тактика за ограничаване на опциите.

ТАКТИКИ ЗА ИЗМЕНЯЕМОСТ – ПРЕДОТВРАТЯВАНЕ НА ЕФЕКТА НА ВЪЛНАТА

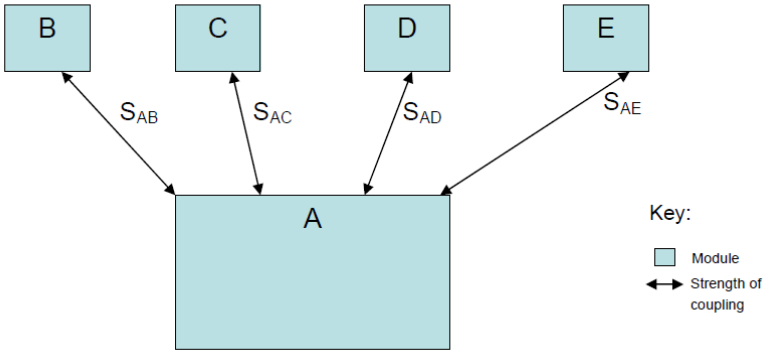
Предотвратяване на ефекта на вълната

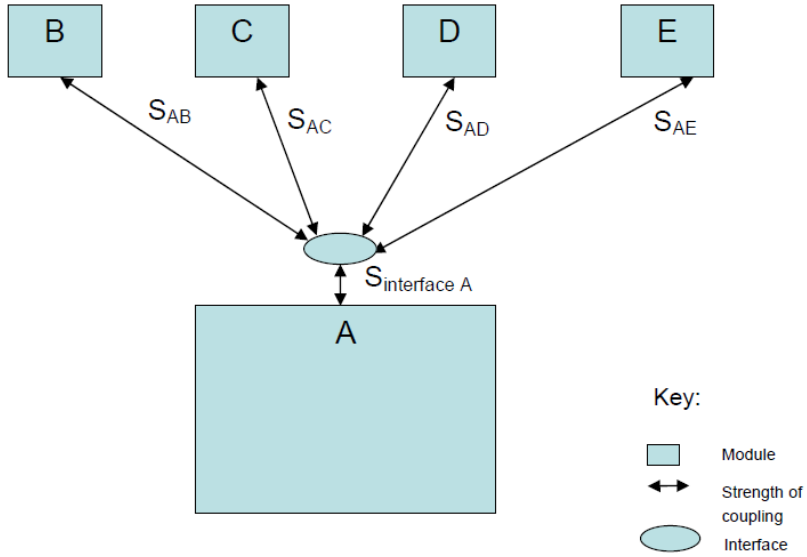
- Ефект на вълната има тогава, когато се налагат модификации в модули, които не са директно засегнати от дадена промяна.
- Напр., ако *модул А* се модифицира, за да се реализира някаква промяна и се налага модификацията на *модул Б* само защото *модул А* е променен. В този смисъл *Б* зависи от *А*.

Предотвратяване на ефекта на вълната

- ***Скриване на информация*** – декомпозиция на отговорността на даден елемент (система или конкретен модул) и възлагането ѝ на по-малки елементи, като при това част от информацията остава публична и част от нея се скрива.
- Публичната функционалност и данни са достъпни посредством специално дефинирани за целта интерфейси.
- Това е най-старата и изпитана техника за ограничаване на промените и е пряко свързана с “очакване на промените”, тъй като именно списъка с очакваните промени е водещ при съставянето на декомпозицията, така че промените да бъдат сведени в рамките на отделни модули.

Скриване на информация



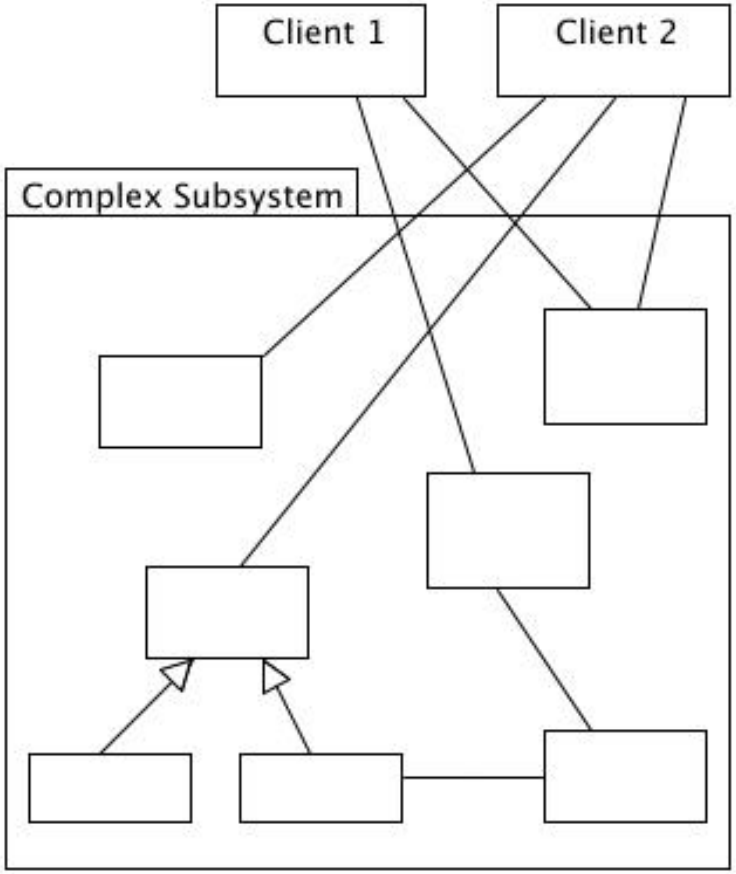


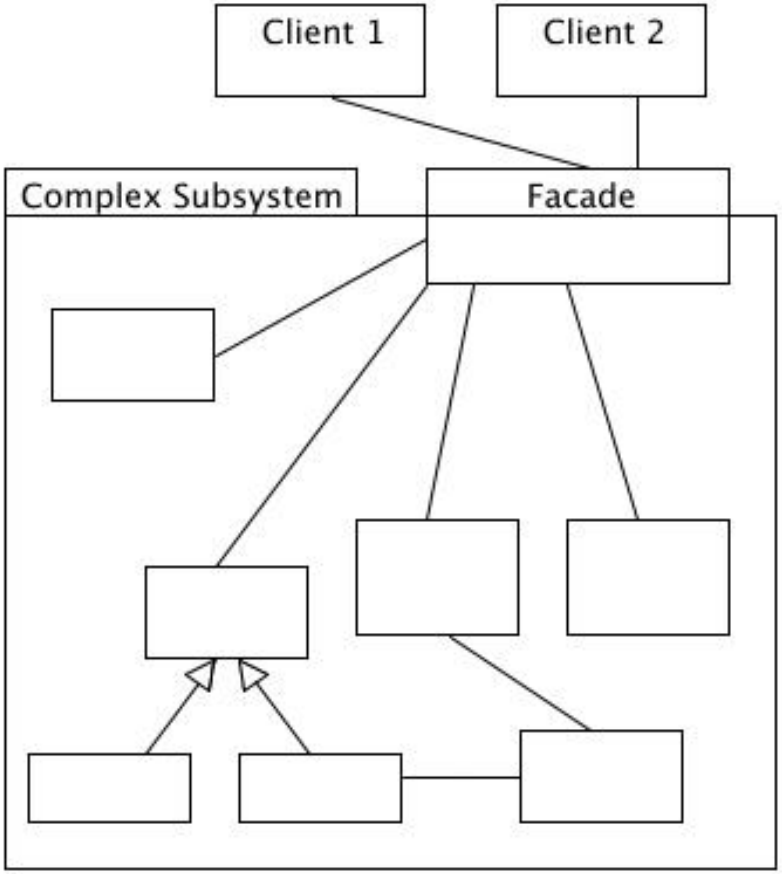
Bachmann, F., L. Bass and R. Nord. Modifiability Tactics. SEI Technical Report. September 2007

Предотвратяване на ефекта на вълната

- ***Ограничаване на комуникацията*** чрез ограничаването на модулите, с които даден модул обменя информация (доколкото това е възможно)
- Т.е. – ограничават се модулите, които консумират данни, създадени от модул А и се ограничават модулите, които създават информация, която се използва от модул А.

Facade





Source: http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/facade.html

Предотвратяване на ефекта на вълната

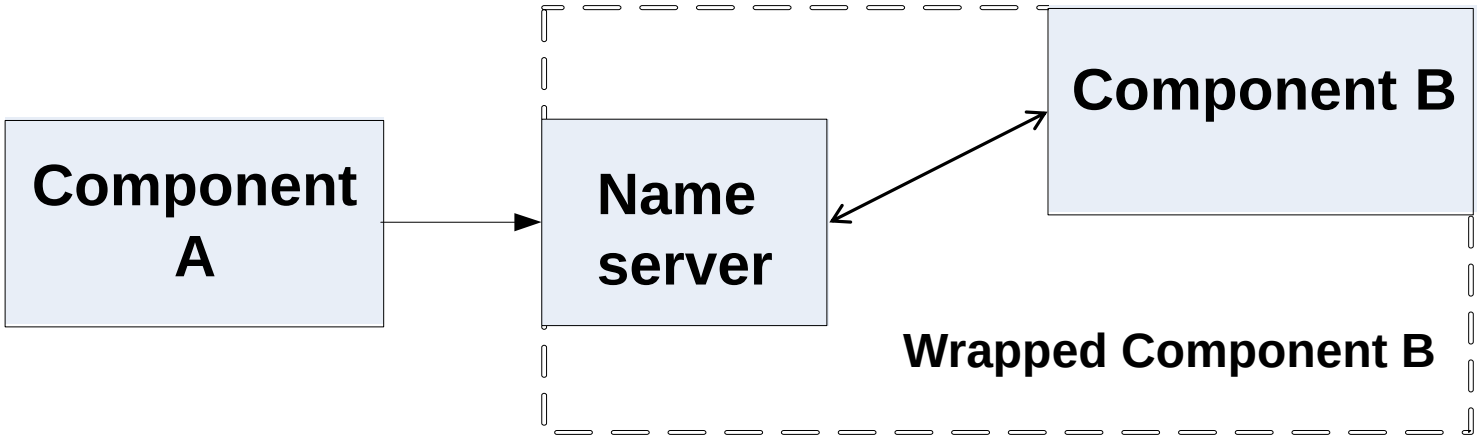
- ***Поддръжка на съществуващите интерфейси*** – Ако Б зависи от името и сигнатурата на даден интерфейс от А, то съответният синтаксис трябва да се поддържа непроменен. За целта се прилагат следните техники:
  - Добавяне на нов интерфейс – вместо да се сменя интерфейс се добавя нов;
  - Добавя се адаптер – А се променя и същевременно се добавя адаптер, чрез който се експлоатира стария синтаксис;
  - Създава се stub – ако се налага А да се премахне, на негово място се оставя stub – процедура със същия синтаксис, която обаче не прави нищо (NOOP);
  - И т.н.

Предотвратяване на ефекта на вълната

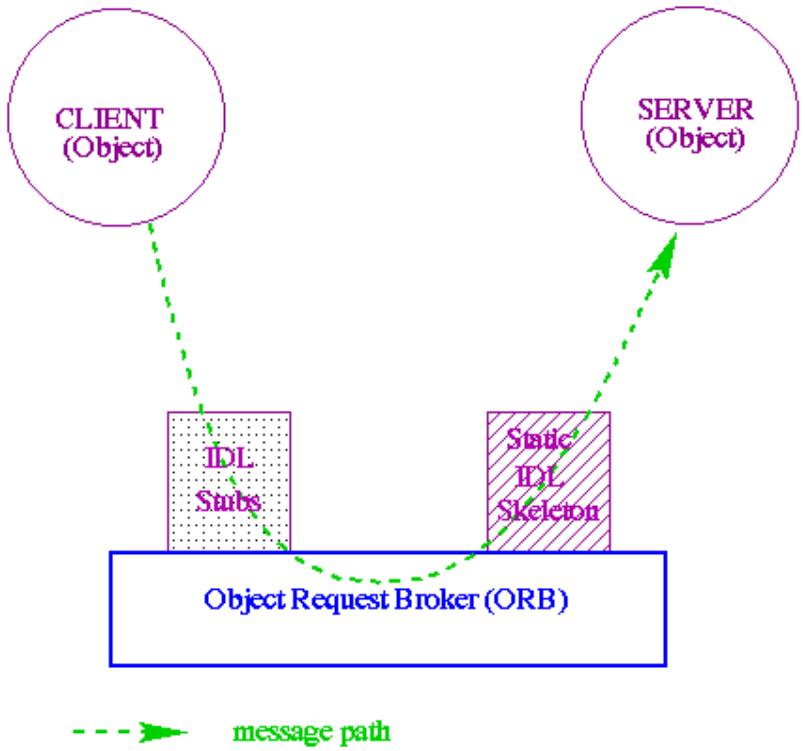
- ***Използване на посредник*** – Ако Б зависи по някакъв начин от А (освен семантично), е възможно между А и Б да бъде поставен „*посредник*“, които премахва тази зависимост.
  - Посредник – wrapper, mediator, façade, adaptor, proxy и т.н....

Name server

- When the client (A) does not know about the location of the server (B), then the wrapper is called “name server”

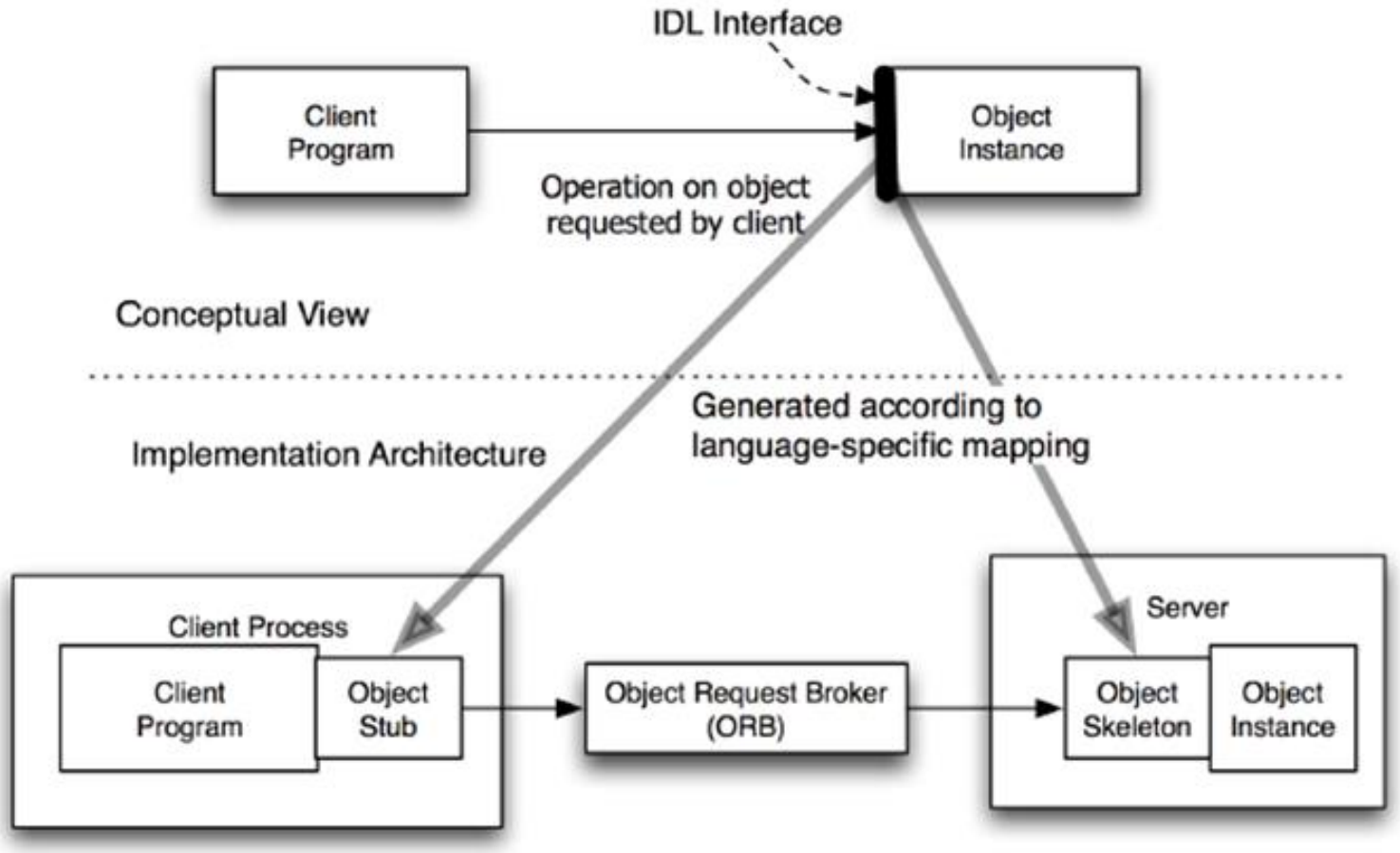


Object request broker



- Very popular for distributed object-oriented systems about a decade ago
- Combines many features of different wrapper implementations

Source: http://www.erlang.org/doc/apps/orber/ch_orber_kernel.html


IDL Interface
Client Program
Object Instance
Operation on object requested by client
Conceptual View
Implementation Architecture
Generated according to language-specific mapping
Client Process
Client Program
Object Stub
Object Request Broker (ORB)
Server
Object Skeleton
Object Instance

Object request broker

- Manages the communication between objects
- Has information about all object in the system and their interfaces
- The so-called *stub* defines the interface of the object being called, while the *skeleton* defines the connection with the caller
- An *Interface Definition Language* (IDL) is used to define the stub and the skeleton

Object request broker

- Industrial specifications of object request broker, and all its environment are called *middleware*
- Middleware provides the following:
  - Interaction support, where the middleware coordinates interactions between different components in the system
    - The middleware provides location transparency in that it isn't necessary for components to know the physical locations of other components.
  - The provision of common components, where the middleware provides reusable implementations of components that may be required by several components in the distributed system.
    - By using these common components, components can easily inter-operate and provide user components in a consistent way.

Industrial middleware specifications

- CORBA (Common Object Request Broker Architecture) – by *Object Management Group* (*OMG*), i.e. it is Java based
- COM (Component Object Model)/DCOM (Distributed COM) by Microsoft, i.e. it is C++ based

ТАКТИКИ ЗА ИЗМЕНЯЕМОСТ – ОТЛАГАНЕ НА СВЪРЗВАНЕТО

Отлагане на свързването

- Двете категории тактики, които разгледахме до тук служат за намаляване на броя на модулите, които подлежат на модификации при нуждата от промяна.
- Само че сценариите за изменяемост включват и елементи, свързани с възможността промени да се правят от не-програмисти, което няма нищо общо с брой модули и т.н.
- За да бъдат възможни тези сценарии се налага да се инвестира в допълнителна инфраструктура, която да позволява именно тази възможност – т.н. отлагане на свързването.

Отлагане на свързването

- Различни “решения” могат да бъдат “свързани” в изпълняваната система по различно време.
- Когато свързването става по време на програмирането, промяната следва да бъде изкомуникирана с разработчика, след което тя да бъде реализирана, да се тества и внедри, и всичко това отнема време.
- От друга страна, ако свързването става по време на зареждането и/или изпълнението, това може да се направи от потребителя/администратора, без намесата на разработчика.
- Необходимата за целта инфраструктура (за извършване на промяната и след това нейното тестване и внедряване) трябва да е вградена в самата система.

Отлагане на свързването

- Съществуват много тактики за отлагане на свързването, по-важните от които са:
  - Включване/изкл./замяна на компоненти, както по време на изпълнението (plug-and-play), така и по време на зареждането (component replacement)
  - Конфигурационни файлове, в които се задават стойностите на различни параметри
  - Дефиниране и придържане към протоколи, които позволяват промяна на компоненти по време на изпълнение

Тактики за сигурност (*Security)*

- Тактиките за сигурност могат да бъдат разделени на:
  - Тактики за устояване на атаките (ключалка на вратата)
  - Тактики за откриване на атаките (аларма)
  - Тактики за възстановяване след атака (застраховка)

Тактики за сигурност – устояване на атаки

- Автентикация на потребителите – проверка за това, дали потребителя е този, за който се представя (пароли, сертификати, биометрика и т.н.)
- Оторизация на потребителите – проверка за това дали потребителя има достъп до определени ресурси
- Конфиденциалност на данните – посредством криптиране (на комуникационните канали и на постоянната памет)
- Интегритет – включване на различни механизми на излишък – чек-суми, хеш-алгоритми и т.н.
- Ограничаване на експозицията, т.е. на местата, чрез които можем да бъдем атакувани
- Ограничаване на достъпа – firewall, DMZ, и др., вкл. и средства за ограничаване на физическия достъп

Тактики за сигурност – възстановяване след атака

- Тактиките за възстановяване след атака са свързани с възстановяване на състоянието и с идентификация на извършителя.
- Тактиките за възстановяване на системата донякъде се препокриват с тактиките за Изправност, тъй като извършена атака може да се разгледа като друг срив в работата на системата. Трябва да се внимава обаче в детайли като пароли, списъци за достъп, потребителски профили и т.н.

Тактики за сигурност – възстановяване след атака

- **Тактиката за идентифициране на атакуващия** е поддръжка на audit trail – копие на всяка транзакция, извършена върху данните, заедно с информация, която идентифицира извършителя по недвусмислен начин. Audit Trail-а може да се използва да се проследят действията на извършителя, с цел осигуряване на невъзможност за отричане (non-repudiation), а също и за възстановяване от атаката.
- Важно е да се отбележи, че audit trail-овете също са обект на атака, така че при проектирането на системата трябва да се вземат мерки достъп до тях да се осигурява само при определени условия.

Тактики за изпитаемост (*Testability*)

- Целта на тактиките за изпитаемост е, да подпомогнат тестването, когато част от софтуерната разработка е приключила
- За фазата преди приключването на разработката за това обикновено се прилагат тактики като *code review*
- За тестване на работеща система обикновено се използва софтуер, който чрез изпълнението на специализирани скриптове подава входни данни на системата и анализира резултата от нейната работа
- Целта на тактиките за изпитаемост е да подпомагат този процес

Тактики за изпитаемост

- Запис и възпроизвеждане – прихващане на информацията, която преминава през даден интерфейс.

  Може да се използва както за генериране на входни данни, така и за запис на изходно състояние с цел последващо сравнение.

Тактики за изпитаемост

- Разделяне на интерфейса от реализацията – позволява замяна на реализацията за тестови цели

Тактики за изпитаемост

- Специализиран интерфейс за тестване – предоставяне на интерфейс за специализиран тестващ софтуер, който е различен от нормалния. Това дава възможност различни мета-данни да бъдат предоставени на тестващия софтуер, които да го управляват и т.н.
- Тестващият софтуер може да тества и тестовия интерфейс, чрез сравнение с резултатите, постигнати чрез използване на нормалния интерфейс.

Тактики за изпитаемост

- Вградени модули за мониторинг – Самата система поддържа информация за състоянието, натовареността, производителността, сигурността и т.н. и я предоставя на определен интерфейс. Интерфейсът може да е постоянен или временен, включен посредством техника за инструментиране.

Тактики за използваемост (*Usability*)

- Използваемостта се занимава с това, колко лесно даден потребител успява да свърши дадена задача и доколко системата подпомага това му действие.
- Различаваме два вида тактики за използваемост, всяка от тях насочена към различна категория “потребители”.
  - Тактики за използваемост по време на изпълнението (*runtime*) – насочени към крайния потребител
  - Тактики за използваемост, свързани с UI – насочени към разработчика на интерфейса. Този вид тактики са силно свързани с тактиките за изменяемост
    - Разделяне на интерфейса от реализацията, например при MVC

Тактики за използваемост – по време на изпълнението (1)

- По време на работа на системата, използваемостта обикновено е свързана с предоставянето на достатъчно информация, обратна връзка и възможност за изпълнение на конкретни команди (cancel, undo, aggregate, multiple views и т.н.)
- В областта на Human-Computer Interaction (HCI) обикновено се говори за “потребителска инициатива”, “системна инициатива” или “смесена инициатива”.
- Напр., когато се отказва дадена команда, потребителя извършва “cancel” (потребителска инициатива) и системата отговаря. По време на извършването на това действие обаче, системата предоставя индикатор за прогреса (системна инициатива).

Тактики за използваемост – по време на изпълнението (2)

- Когато става въпрос за потребителска инициатива, архитектът проектира реакцията на системата както при всички останали функционалности, т.е. трябва да се опише поведението на системата в зависимост от получените входни събития. Напр., при cancel:
  - Системата трябва да е подготвена за cancel (т.е. да има процес, който следи за такава команда, който да не се блокира от действието на процеса, който се отказва);
  - Процесът, който се отказва трябва да се преустанови;
  - Ресурсите, които са били заети да се освободят;
  - Евентуално компонентите, които взаимодействат с отказания процес да се уведомят, за да предприемат съответните действия;

Тактики за използваемост – по време на изпълнението (3)

- Когато става въпрос за системна инициатива, системата трябва да разчита на някаква информация (модел) относно потребителя, задачата или моментното си състояние.
- Различните модели изискват различни входни данни за постигането на инициативата. Тактиките за постигането на използваемост по време на системната инициатива са свързани с избора на конкретен модел.

Тактики за използваемост – по време на изпълнението (4)

- Моделиране на задачата – системата знае какво прави потребителя и може да му помогне в зависимост от създадения модел (напр. Auto-Correction);
- Моделиране на потребителя – моделът съдържа информация за това, както потребителя знае за системата, как работи с нея, какво очаква и т.н. Това позволява на системата да бъде пригодена към поведението на потребителя.
- Моделиране на системата – моделът на системата включва очакваното поведение, така че тя да предоставя адекватна обратна връзка. (напр. предвиждане на времето за обработка, за да бъде показан progress bar).